「2000年記念並列処理シンポジウム JSPP2000」 平成12年5月

分散メモリ型並列計算機HITACHI SR8000における並列FFTアルゴリズム

高 橋 大 介[†,☆]

本論文では，ベクトルSMPノードを搭載した分散メモリ型並列計算機における並列一次元FFTアルゴリズムを提案する．four-step FFTアルゴリズムは最内側ループ長を長くするために，five-step FFTアルゴリズムに拡張することができる．並列一次元FFTアルゴリズムの実現に際しては，four-step FFTおよびfive-step FFTアルゴリズムを用いた．本論文で提案する並列一次元FFTアルゴリズムでは，データ配置にサイクリック分割を適用することにより，入力と出力を同じデータ配置にした場合でも全対全通信が1回で済む．提案するアルゴリズムに基づいて，並列一次元FFTを(疑似)ベクトルノードを搭載した分散メモリ型並列計算機HITACHI SR8000に実現し，性能評価を行った．その結果，16ノードのSR8000では約38 GFLOPSの性能を得ることができた．

Parallel FFT Algorithms for the Distributed-Memory Parallel Computer HITACHI SR8000

DAISUKE TAKAHASHI[†,☆]

In this paper, we propose high-performance parallel one-dimensional fast Fourier transform (FFT) algorithms for distributed-memory parallel computers with vector symmetric multiprocessor (SMP) nodes. The four-step FFT algorithm can be altered into a five-step FFT algorithm to expand the innermost loop length. We use the four-step and five-step FFT algorithms to implement the parallel one-dimensional FFT algorithms. In our proposed parallel one-dimensional FFT algorithms, since we use cyclic distribution, all-to-all communication takes place only once. Moreover, the input data and output data are both in natural order. Performance results of one-dimensional FFTs on a distributed-memory parallel computer with (pseudo) vector SMP nodes, HITACHI SR8000, are reported. We succeeded in obtaining performance of about 38 GFLOPS on a 16-node SR8000.

1. はじめに

高速Fourier変換（fast Fourier transform，以下FFT）[1]は，科学技術計算において今日広く用いられているアルゴリズムである．FFTにおいて大量のデータを高速に処理するために，分散メモリ型並列計算機におけるFFTアルゴリズム[2)~7)]が多く提案されている．

近年，SMP（Symmetric Multi-Processor）構成のノードを多数結合した分散メモリ型並列計算機が普及しつつある．SMPノードを搭載した分散メモリ型並列計算機は，ノード内の共有メモリアーキテクチャと，ノード間の分散メモリアーキテクチャの両方の性質を持つ．したがって，SMPノードを搭載した分散メモリ型並列計算機の性能を最大限に引き出すためには，これらのアーキテクチャの性質を生かしたアルゴリズムを構築することが重要である．

これまでに，分散メモリ型並列計算機の各ノードがスカラープロセッサによるSMP構成である場合については，並列FFTアルゴリズムが提案されている[8)]が，各ノードがベクトルプロセッサによるSMP構成の場合については実現や評価が十分に行われていないのが現状である．

そこで本論文では，ベクトルSMPノードを搭載した分散メモリ型並列計算機において並列一次元FFTを実現し，その性能を評価した結果を述べる．

ベクトルSMPノードを搭載した分散メモリ型並列計算機を分散メモリアーキテクチャの観点から考えた場合，高い性能を得るためには，各ノードのロードバランスを均一にするとともに，ノード間の通信量および通信回数をできるだけ減らすことが必要になる．また，ノード内の共有メモリアーキテクチャの観点からは，ベクトルSMPノード内において高い性能を得る

† 東京大学情報基盤センター
Information Technology Center, University of Tokyo
☆ 現在，埼玉大学大学院理工学研究科
Presently with Graduate School of Science and Engineering, Saitama University

ためには，最内側ループ長をできるだけ長くするとともに，ノード内における並列性を確保することが重要である．

従来の four-step FFT アルゴリズム [9] において，さらに最内側ループ長を長くするために，本論文では five-step FFT を提案し，four-step FFT および five-step FFT アルゴリズムに基づき二種類の並列一次元 FFT アルゴリズムを構成できることを示す．

多くの並列一次元 FFT アルゴリズムにおいては，全対全通信を用いてノード間でデータを入れ換える必要があることが知られている [4)~7)]．全対全通信は，分散メモリ型並列計算機ではコストの高い通信であるので，回数をできるだけ減らす必要がある．

この全対全通信に関しては，入力のデータ配置を転置したものを出力とすることにより，全対全通信の回数を 1 回にする手法 [4] が提案されているが，この手法では入力と出力を同じデータ配置にする必要がある場合には，全対全通信が 2 回必要となる．

全対全通信の回数を減らすために，Edelman らは fast multipole method[10),11)] を用いて仮想的にデータを入れ換えることにより，入力と出力を同じデータ配置にした場合でも全対全通信の回数を 1 回にする手法を提案している [7]．

本論文では，データ分割にサイクリック分割を適用するとともに，ノード内で行列の転置を行ってから全対全通信を行い，その後にノード内で行列の再配置を行うことにより，Edelman らの手法のような複雑な処理を行わずに，入力と出力を同じデータ配置にした場合でも全対全通信の回数を 1 回に減らす手法を示す．

これらの並列一次元 FFT アルゴリズムを（疑似）ベクトル SMP ノードを搭載した分散メモリ型並列計算機 HITACHI SR8000 に実現し，性能評価を行う．

以下，2 章で HITACHI SR8000 について簡単に説明する．3 章で four-step FFT アルゴリズムについて，4 章で提案する five-step FFT アルゴリズムについて，5 章で並列一次元 FFT アルゴリズムについて述べる．6 章でベクトル SMP ノード内における FFT アルゴリズムについて述べる．7 章で本論文で示す並列一次元 FFT アルゴリズムの性能評価結果を示す．最後の 8 章はまとめである．

2. HITACHI SR8000

HITACHI SR8000 は，（疑似）ベクトル SMP ノードを搭載した分散メモリ型並列計算機である．SR8000 は 4 個 ~128 個のノードから構成される．各ノードは疑似ベクトル処理機構 [12] を備えた 8 個の RISC プロセッサから成っており，理論ピーク性能は 8 GFLOPS，最大メモリ容量は 8 GB である．SR8000 の各ノードは多次元クロスバネットワークで接続されており，ノード間最大通信速度は 1 GB/秒（片方向），2 GB/秒（双方向）である．

3. Four-Step FFT アルゴリズム

FFT は，離散 Fourier 変換（discrete Fourier transform，以下 DFT）を高速に計算するアルゴリズムとして知られている．DFT は次式で定義される．

$$y_k = \sum_{j=0}^{n-1} x_j \omega_n^{jk}, \quad 0 \le k \le n-1 \qquad (1)$$

ここで，$\omega_n = e^{-2\pi i/n}$，$i = \sqrt{-1}$ である．

$n = n_1 \times n_2$ と分解できるものとすると，式 (1) における j および k は，

$$j = j_1 + j_2 n_1, \quad k = k_2 + k_1 n_2 \qquad (2)$$

と書くことができる．そのとき，式 (1) の x と y は次のような二次元配列（columnwise）で表すことができる．

$$x_j = x(j_1, j_2), \quad 0 \le j_1 \le n_1 - 1, \quad 0 \le j_2 \le n_2 - 1 \qquad (3)$$

$$y_k = y(k_2, k_1), \quad 0 \le k_1 \le n_1 - 1, \quad 0 \le k_2 \le n_2 - 1 \qquad (4)$$

したがって，式 (1) は式 (5) のように変形できる．

$$y(k_2, k_1) = \sum_{j_1=0}^{n_1-1} \sum_{j_2=0}^{n_2-1} x(j_1, j_2) \omega_{n_2}^{j_2 k_2} \omega_{n_1 n_2}^{j_1 k_2} \omega_{n_1}^{j_1 k_1} \qquad (5)$$

式 (5) から次に示されるような，four-step FFT アルゴリズム [9] が導かれる．

Step 1: n_1 組の n_2 点 multirow FFT

$$x_1(j_1, k_2) = \sum_{j_2=0}^{n_2-1} x(j_1, j_2) \omega_{n_2}^{j_2 k_2}$$

Step 2: ひねり係数の乗算

$$x_2(j_1, k_2) = x_1(j_1, k_2) \omega_{n_1 n_2}^{j_1 k_2}$$

Step 3: 転置

$$x_3(k_2, j_1) = x_2(j_1, k_2)$$

Step 4: n_2 組の n_1 点 multirow FFT

$$y(k_2, k_1) = \sum_{j_1=0}^{n_1-1} x_3(k_2, j_1) \omega_{n_1}^{j_1 k_1}$$

Step 2 における $\omega_{n_1 n_2}^{j_1 k_2}$ は，ひねり係数 [13] と呼ばれる 1 の原始根であり，複素数である．

この four-step FFT アルゴリズムの特徴を以下に示す．

- $n_1 = n_2 = \sqrt{n}$ とした場合，$\sqrt{n}$ 組の $\sqrt{n}$ 点 multirow FFT[14] が Step 1 と 4 で行われる．したがって，最内側ループ長は $\sqrt{n}$ に固定されるので，このアルゴリズムはベクトルプロセッサに適している．
- 行列の転置が 1 回で済む．

4. Five-Step FFT アルゴリズム

3 章で述べた従来の four-step FFT アルゴリズムにおいて，最内側ループ長をさらに長くするために，four-step FFT アルゴリズムにおける二次元表現を三次元表現に拡張することを考える．

n が $n = n_1 \times n_2 \times n_3$ と分解できるものとすると，式 (1) における j および k は，

$$\begin{aligned} j &= j_1 + j_2 n_1 + j_3 n_1 n_2 \\ k &= k_3 + k_2 n_3 + k_1 n_2 n_3 \end{aligned} \qquad (6)$$

と書くことができる．そのとき，式 (1) の x と y は次のような三次元配列 (columnwise) で表すことができる．

$$x_j = x(j_1, j_2, j_3), \quad 0 \le j_1 \le n_1 - 1, \quad 0 \le j_2 \le n_2 - 1, \quad 0 \le j_3 \le n_3 - 1 \quad (7)$$

$$y_k = y(k_3, k_2, k_1), \quad 0 \le k_1 \le n_1 - 1, \quad 0 \le k_2 \le n_2 - 1, \quad 0 \le k_3 \le n_3 - 1 \quad (8)$$

したがって，式 (1) は式 (9) のように変形できる．

$$y(k_3, k_2, k_1) = \sum_{j_1=0}^{n_1-1} \sum_{j_2=0}^{n_2-1} \sum_{j_3=0}^{n_3-1} x(j_1, j_2, j_3) \omega_{n_3}^{j_3 k_3} \omega_{n_2 n_3}^{j_2 k_3} \omega_{n_2}^{j_2 k_2} \omega_n^{j_1 k_3} \omega_{n_1 n_2}^{j_1 k_2} \omega_{n_1}^{j_1 k_1} \quad (9)$$

式 (9) から次に示されるような，five-step FFT アルゴリズムが導かれる．

Step 1: $n_1 n_2$ 組の n_3 点 multirow FFT

$$x_1(j_1, j_2, k_3) = \sum_{j_3=0}^{n_3-1} x(j_1, j_2, j_3) \omega_{n_3}^{j_3 k_3}$$

Step 2: ひねり係数の乗算および転置

$$x_2(k_3, j_1, j_2) = x_1(j_1, j_2, k_3) \omega_{n_2 n_3}^{j_2 k_3}$$

Step 3: $n_3 n_1$ 組の n_2 点 multirow FFT

$$x_3(k_3, j_1, k_2) = \sum_{j_2=0}^{n_2-1} x_2(k_3, j_1, j_2) \omega_{n_2}^{j_2 k_2}$$

Step 4: ひねり係数の乗算および再配置

$$x_4(k_3, k_2, j_1) = x_3(k_3, j_1, k_2) \omega_n^{j_1 k_3} \omega_{n_1 n_2}^{j_1 k_2}$$

Step 5: $n_3 n_2$ 組の n_1 点 multirow FFT

$$y(k_3, k_2, k_1) = \sum_{j_1=0}^{n_1-1} x_4(k_3, k_2, j_1) \omega_{n_1}^{j_1 k_1}$$

この five-step FFT アルゴリズムの特徴を以下に示す．

- $n_1 = n_2 = n_3 = n^{1/3}$ とした場合，$n^{2/3}$ 組の $n^{1/3}$ 点 multirow FFT が Step 1, 3 と 5 で行われる．したがって，最内側ループ長は $n^{2/3}$ に固定されるので，このアルゴリズムは 3 章で述べた従来の four-step FFT アルゴリズムよりベクトルプロセッサに適している．
- 行列の転置および再配置がそれぞれ 1 回必要となる．

5. 並列一次元 FFT アルゴリズム

5.1 Four-Step FFT に基づく並列一次元 FFT アルゴリズム

並列一次元 FFT アルゴリズムとしては，four-step FFT の変形である six-step FFT[9] に基づく並列一次元 FFT アルゴリズム [14] が知られているが，最内側ループ長が短いために，分散メモリ型並列計算機の各ノードがベクトルプロセッサの場合には適していないことが指摘されている [6]．

そこで本節では並列一次元 FFT アルゴリズムを考えるにあたって four-step FFT の考え方を適用する．

一次元 FFT においてデータ数 N が $N = N_1 \times N_2$ と分解されるとする．すると，一次元配列 $x(N)$ は二次元配列 $x(N_1, N_2)$ と表すことができる．P 台のノードを持つ分散メモリ型並列計算機では，この配列 $x(N_1, N_2)$ は一次元目 (N_1) に沿って分散される．N_1 が P で割り切れるとすると，各ノードには N/P 個のデータが分散されることになる．

やや複雑になるが，ここで $\hat{N}_r \equiv N_r/P$ の記法を導入する．そして，インデックス J_r に沿ったデータがすべての P 台のノードに分散されることを示す記法を $\hat{J}_r$ とする．なお，r は次元 r にインデックスが属しているという意味である．

これより，分散された二次元配列は $\hat{x}(\hat{N}_1, N_2)$ と表すことができる．サイクリック分割によると，m 番目のノードにおけるローカルインデックス $\hat{J}_r(m)$ は，次のようなグローバルインデックス J_r に一致する．

$$J_r = \hat{J}_r(m) \times P + m, \quad 0 \le m \le P - 1, \quad 1 \le r \le 2 \qquad (10)$$

ここで全対全通信を示すために，$\tilde{N}_i \equiv N_i/P_i$ の記法を導入する．この記法を用いると，N_i は $\tilde{N}_i$ と P_i の二次元表現に分解される．なお，P_i は P と同じものを示しているが，このインデックスが次元 i に属していることを示している．

初期データを $\hat{x}(\hat{N}_1, N_2)$ とすると，four-step FFT に基づく並列一次元 FFT アルゴリズムは次のようになる．

Step 1: N_1/P 組の N_2 点 multirow FFT

$$\hat{x_1}(\hat{J}_1, K_2) = \sum_{J_2=0}^{N_2-1} \hat{x}(\hat{J}_1, J_2) \omega_{N_2}^{J_2 K_2}$$

Step 2: ひねり係数の乗算およびノード内転置

$$\hat{x_2}(P_2, \tilde{K}_2, \hat{J}_1) \equiv \hat{x_2}(K_2, \hat{J}_1) = \hat{x_1}(\hat{J}_1, K_2) \omega_{N_1 N_2}^{\hat{J}_1 K_2}$$

Step 3: ノード内転置

$$\hat{x_3}(\tilde{K}_2, \hat{J}_1, P_2) = \hat{x_2}(P_2, \tilde{K}_2, \hat{J}_1)$$

Step 4: 全対全通信

$$\hat{x}_4(\hat{K}_2,\ \tilde{J}_1,\ P_1) = \hat{x}_3(\tilde{K}_2,\ \hat{J}_1,\ P_2)$$

Step 5: ノード内再配置

$$\hat{x}_5(\hat{K}_2,\ J_1) \equiv \hat{x}_5(\hat{K}_2,\ P_1,\ \tilde{J}_1) = \hat{x}_4(\hat{K}_2,\ \tilde{J}_1,\ P_1)$$

Step 6: N_2/P 組の N_1 点 multirow FFT

$$\hat{y}(\hat{K}_2,\ K_1) = \sum_{J_1=0}^{N_1-1} \hat{x}_5(\hat{K}_2,\ J_1)\omega_{N_1}^{J_1K_1}$$

上記の並列一次元 FFT アルゴリズムにおいて，Step 1 と 6 で multirow FFT が実行される．Step 2 での計算は，ひねり係数の乗算とノード内転置を同時に行うことにより，メモリバンド幅を有効に活用することができる．Step 3 と 5 では，全対全通信を 1 回で済ませるために，ノード内転置およびノード内再配置を行う．Step 4 では全対全通信が行われる．

four-step FFT に基づく並列一次元 FFT アルゴリズムの特徴は，次に示す通りである．

- $N_1 = N_2 = \sqrt{N}$ とした場合，$\sqrt{N}/P$ 組の $\sqrt{N}$ 点 multirow FFT が Step 1 と 6 で実行される．したがって，最内側ループ長は $\sqrt{N}/P$ に固定される．
- 全対全通信が 1 回で済む．しかも，入力データ x と出力データ y は共に正順となる．

5.2 Five-Step FFT に基づく並列一次元 FFT アルゴリズム

5.1 節で述べた four-step FFT に基づく並列一次元 FFT アルゴリズムでは，データ数を N，ノード数を P とした場合，最内側ループ長は $\sqrt{N}/P$ であった．本節では最内側ループ長をさらに長くする方法として，4 章で述べた five-step FFT を並列一次元 FFT アルゴリズムに適用することを考える．

一次元 FFT においてデータ数 N が $N = N_1 \times N_2 \times N_3$ と分解されるとし，P をノード数とする．5.1 節で導入した記法を用いて，初期データを $\hat{x}(\hat{N}_1,\ N_2,\ N_3)$ とすると，five-step FFT に基づく並列一次元 FFT アルゴリズムは次のようになる．

Step 1: $(N_1/P)\cdot N_2$ 組の N_3 点 multirow FFT

$$\hat{x}_1(\hat{J}_1,\ J_2,\ K_3) = \sum_{J_3=0}^{N_3-1} \hat{x}(\hat{J}_1,\ J_2,\ J_3)\omega_{N_3}^{J_3K_3}$$

Step 2: ひねり係数の乗算およびノード内転置

$$\hat{x}_2(K_3,\ \hat{J}_1,\ J_2) = \hat{x}_1(\hat{J}_1,\ J_2,\ K_3)\omega_{N_2N_3}^{J_2K_3}$$

Step 3: $N_3\cdot(N_1/P)$ 組の N_2 点 multirow FFT

$$\hat{x}_3(K_3,\ \hat{J}_1,\ K_2) = \sum_{J_2=0}^{N_2-1} \hat{x}_2(K_3,\ \hat{J}_1,\ J_2)\omega_{N_2}^{J_2K_2}$$

Step 4: ひねり係数の乗算およびノード内再配置

$$\hat{x}_4(P_3, \tilde{K}_3,\ K_2,\ \hat{J}_1) \equiv \hat{x}_4(K_3,\ K_2,\ \hat{J}_1) = \hat{x}_3(K_3,\ \hat{J}_1,\ K_2)\omega_N^{\hat{J}_1(K_3+K_2N_3)}$$

Step 5: ノード内転置

$$\hat{x}_5(\tilde{K}_3,\ K_2,\ \hat{J}_1,\ P_3) = \hat{x}_4(P_3,\ \tilde{K}_3,\ K_2,\ \hat{J}_1)$$

Step 6: 全対全通信

$$\hat{x}_6(\hat{K}_3,\ K_2,\ \tilde{J}_1,\ P_1) = \hat{x}_5(\tilde{K}_3,\ K_2,\ \hat{J}_1,\ P_3)$$

Step 7: ノード内再配置

$$\hat{x}_7(\hat{K}_3,\ K_2,\ J_1) \equiv \hat{x}_7(\hat{K}_3,\ K_2,\ P_1,\ \tilde{J}_1) = \hat{x}_6(\hat{K}_3,\ K_2,\ \tilde{J}_1,\ P_1)$$

Step 8: $(N_3/P)\cdot N_2$ 組の N_1 点 multirow FFT

$$\hat{y}(\hat{K}_3,\ K_2,\ K_1) = \sum_{J_1=0}^{N_1-1} \hat{x}_7(\hat{K}_3,\ K_2,\ J_1)\omega_{N_1}^{J_1K_1}$$

上記の並列一次元 FFT アルゴリズムにおいて，Step 1，3 と 8 で multirow FFT が実行される．Step 2 と 4 での計算は，ひねり係数の乗算とノード内転置（再配置）を同時に行うことにより，メモリバンド幅を有効に活用することができる．Step 5 と 7 では，全対全通信を 1 回で済ませるために，ノード内転置およびノード内再配置を行う．Step 6 では全対全通信が行われる．

five-step FFT に基づく並列一次元 FFT アルゴリズムの特徴は，次に示す通りである．

- $N_1 = N_2 = N_3 = N^{2/3}$ とした場合，$N^{2/3}/P$ 組の $N^{1/3}$ 点 multirow FFT が Step 1，3 と 8 で実行される．したがって，最内側ループ長は $N^{2/3}/P$ に固定される．
- four-step FFT に基づく並列一次元 FFT アルゴリズムと同様に，全対全通信が 1 回で済む．しかも，入力データ x と出力データ y は共に正順となる．

5.3 最内側ループ長の検討

four-step FFT および five-step FFT に基づく並列一次元 FFT アルゴリズムにおいて，各ノード内における最内側ループ長を検討する．four-step FFT に基づく並列一次元 FFT アルゴリズムでは，$N = N_1 \times N_2$ で表される場合に $N_1 = N_2 = \sqrt{N}$ とし，ノード数を P とすると，最内側ループ長は $\sqrt{N}/P$ となる．

一方，five-step FFT に基づく並列一次元 FFT アルゴリズムでは，$N = N_1 \times N_2 \times N_3$ で表される場合に $N_1 = N_2 = N_3 = N^{1/3}$ とすると，最内側ループ長は $N^{2/3}/P$ となる．

例えば，$N = 2^{24}$ として $P = 128$ とした場合，最内側ループ長は four-step FFT に基づく並列一次元 FFT アルゴリズムでは 32 $(= \sqrt{2^{24}}/128)$ であるのに対し，five-step FFT に基づく並列一次元 FFT アルゴリズムでは 512 $(= (2^{24})^{2/3}/128)$ となる．

これらの結果から，最内側ループ長に関しては，five-step FFT に基づく並列一次元 FFT アルゴリズムが four-step FFT に基づく並列一次元 FFT アルゴリズムに比べて有利であることが分かる．

表 1　最内側ループにおける基数 2，4，8 の DIF Stockham アルゴリズムに基づく FFT カーネルの実演算回数（HITACHI SR8000）

	基数 2	基数 4	基数 8
ロード + ストア回数	8	16	32
乗算回数	4	12	32
加算回数	6	22	66
総浮動小数点演算回数 ($n \log_2 n$)	5.000	4.250	4.083
浮動小数点演算命令数	8	28	84
浮動小数点演算命令数と ロード + ストア回数の比	1.000	1.750	2.625

6. ベクトル SMP ノード内における FFT アルゴリズム

共有メモリ型ベクトル並列計算機における FFT アルゴリズムが多く提案されている [2),9),15)]．ベクトル SMP ノード内における FFT アルゴリズムとしては Stockham アルゴリズム [16),17)] に基づく multirow FFT アルゴリズムを用いた．Stockham アルゴリズムは，Cooley-Tukey アルゴリズム [1)] のように入力と出力が重ね書きできないために，入力と出力で別の配列が必要となり，その結果 Cooley-Tukey のアルゴリズムの 2 倍のメモリ容量が必要になる．しかし，ビットリバース処理 [1)] が不要であるためにベクトルプロセッサに適しているアルゴリズムとしても知られている [17)]．また，multirow FFT アルゴリズムでは，複数組の部分を最内側ループで処理することで，最内側ループを長くすることができ，しかも配列アクセスが連続的になるという利点がある．

2 べきの FFT では，基数 2 の FFT に比べて演算回数およびメモリアクセスのより少ない基数 4，8 の FFT [18)] を適用することにより，効率を高くすることができる [14)]．したがって，本論文では基数 2，4，8 の組み合わせで実現および評価を行った．

SR8000 の各ノード内のプロセッサは $a = \pm a \pm bc$ で表されるような 3 オペランドの積和演算命令を持っている．ここで，a，b，c は浮動小数点レジスタである．Goedecker は基数 2，3，4，5 の FFT カーネルにおいて積和演算命令回数を最小にするアルゴリズムを提案している [19)]．また，高橋，金田は基数 8 の FFT カーネルにおいて同様のアルゴリズムを提案している [20)]．

しかし，これらの積和演算命令に向いた FFT カーネルは $d = \pm a \pm bc$ で表されるような 4 オペランドの積和演算命令を持つプロセッサでは有効であるが，3 オペランドの積和演算命令を持つプロセッサでは効果がない．したがって，今回は従来の周波数間引き（decimation-in-frequency，以下 DIF）の Stockham アルゴリズムを用いた．

表 1 は最内側ループにおける基数 2，4，8 の DIF Stockham アルゴリズムに基づく FFT カーネルの実演算回数を示している．表 1 における浮動小数点演算命令数とは，浮動小数点の乗算，加算および積和演算をそれぞれ 1 命令とした場合の演算命令数である．FFT カーネル内において，積和演算命令が適用できる部分があるために，浮動小数点演算命令数は乗算回数と加算回数の合計より少なくなっていることに注意する．

表 1 から分かるように，高い基数の FFT カーネルはロード + ストア回数の面からも演算回数の面からも有利であることが分かる．さらに，浮動小数点演算命令数とロード + ストア回数の比は，基数 8 の FFT は基数 2 の FFT に比べて 2.625 倍であり，基数 4 の FFT に比べても 1.5 倍となっている．これは，基数を大きくするに従ってデータを再利用できる回数が増えるためにロードとストアの回数が減るからである [14)]．

2 点 FFT を除く 2 べきの FFT では，基数 4 と基数 8 の組み合わせにより FFT を計算し，基数 2 の FFT カーネルを排除することにより，ロードとストア回数および演算回数を減らすことができ，より高い性能を得ることができる．具体的には，$n = 2^p$ $(p \geq 2)$ 点 FFT を $n = 4^q 8^r$ $(0 \leq q \leq 2,\ r \geq 0)$ として計算することにより，基数 4 と基数 8 の FFT カーネルのみで $n \geq 4$ の場合に 2 べきの FFT を計算することができる．

このように，ベクトル SMP ノード内における FFT アルゴリズムでは，ノード内のメモリアクセス回数を減らす工夫が必要となる．

基数 4 および基数 8 の DIF Stockham アルゴリズムに基づく ns 組の n 点 multirow FFT をそれぞれ**図 1**，**図 2** に示す．図 1 および図 2 に示した基数 4，8 の multirow FFT では，最内側ループで ns 組の n 点 FFT が同時に処理されていることが分かる．

6.1 ベクトル SMP ノード内における並列性

SR8000 の各ノードでは，FFT カーネルの外側のループが各ノードの 8 個のプロセッサに分配される．図 1 および図 2 で示した基数 4，8 の FFT カーネルでは，**do** j ループがこれに相当する．

ところが，基数 4，8 の FFT カーネルのどちらにおいても，**do** t ループの最後の反復である $t = p$ の場合には，**do** j ループのループ長 l が 1 となってしまうために，各ノードの 8 個のプロセッサにループを

```
n = 4^p, ω_q = e^{-2πi/q}
do t = 1, p
  l = 4^{p-t}, m = 4^{t-1}

  complex*16 X_{t-1}(ns, m, 4*l), X_t(ns, 4*m, l)
  do j = 1, l
    do k = 1, m
      do row = 1, ns
        c_0 = X_{t-1}(row, k, j)
        c_1 = X_{t-1}(row, k, j+l)
        c_2 = X_{t-1}(row, k, j+2*l)
        c_3 = X_{t-1}(row, k, j+3*l)
        d_0 = c_0 + c_2
        d_1 = c_0 - c_2
        d_2 = c_1 + c_3
        d_3 = -i(c_1 - c_3)
        X_t(row, k, j)       = d_0 + d_2
        X_t(row, k+m, j)     = ω_{4l}^{j-1}(d_1 + d_3)
        X_t(row, k+2*m, j)   = ω_{4l}^{2(j-1)}(d_0 - d_2)
        X_t(row, k+3*m, j)   = ω_{4l}^{3(j-1)}(d_1 - d_3)
      end do
    end do
  end do
end do
```

図 1　基数 4 の DIF Stockham アルゴリズムに基づく ns 組の n 点 multirow FFT

分配することができず，ノード内のプロセッサ利用率が悪くなる．

したがって，$l < 8$ の場合には，**do** j と **do** k のループを入れ換え，**do** k ループを各ノードの 8 個のプロセッサに分配することにより，プロセッサ利用率を向上させることができる．

このように，ベクトル SMP ノードでは，最内側ループ長だけでなく，外側ループ長も考慮する必要がある．

以上のような工夫を用いても，外側ループ長が短い場合には，リストベクトルを用いて **do** j と **do** k の二重ループを一重ループにするか，最内側ループを各ノードのプロセッサに分配する方法が考えられる．

7. 性能評価

並列一次元 FFT の性能評価にあたっては，$N = 2^m$ の m およびノード数 P を変化させて順方向 N 点 FFT を 10 回実行し，その平均の経過時間を測定した．なお，FFT の計算は倍精度複素数で行い，三角関数のテーブルはあらかじめ作り置きとしている．

（疑似）ベクトル SMP ノードを搭載した分散メモリ型並列計算機として，HITACHI SR8000（128 ノード，総メモリ容量 1024 GB，理論ピーク性能 1024 GFLOPS）のうち，1 ノード ～16 ノードを用いた．

7.1 HITACHI SR8000 による測定結果

SR8000 のノード間通信ライブラリとして，MPI を用いた．プログラムはすべて FORTRAN で記述した．コンパイラは日立の最適化 FORTRAN77 V01-00 を用い，最適化オプションとして`-nolimit -rdma -W0,'opt(o(ss))'` を指定した☆．

```
n = 8^p, ω_q = e^{-2πi/q}
do t = 1, p
  l = 8^{p-t}, m = 8^{t-1}

  complex*16 X_{t-1}(ns, m, 8*l), X_t(ns, 8*m, l)
  do j = 1, l
    do k = 1, m
      do row = 1, ns
        c_0 = X_{t-1}(row, k, j)
        c_1 = X_{t-1}(row, k, j+l)
        c_2 = X_{t-1}(row, k, j+2*l)
        c_3 = X_{t-1}(row, k, j+3*l)
        c_4 = X_{t-1}(row, k, j+4*l)
        c_5 = X_{t-1}(row, k, j+5*l)
        c_6 = X_{t-1}(row, k, j+6*l)
        c_7 = X_{t-1}(row, k, j+7*l)
        d_0 = c_0 + c_4
        d_1 = c_0 - c_4
        d_2 = c_2 + c_6
        d_3 = -i(c_2 - c_6)
        d_4 = c_1 + c_5
        d_5 = c_1 - c_5
        d_6 = c_3 + c_7
        d_7 = c_3 - c_7
        e_0 = d_0 + d_2
        e_1 = d_0 - d_2
        e_2 = d_4 + d_6
        e_3 = -i(d_4 - d_6)
        e_4 = (√2/2)(d_5 - d_7)
        e_5 = -(√2/2)i(d_5 + d_7)
        e_6 = d_1 + e_4
        e_7 = d_1 - e_4
        e_8 = d_3 + e_5
        e_9 = d_3 - e_5
        X_t(row, k, j)       = e_0 + e_2
        X_t(row, k+m, j)     = ω_{8l}^{j-1}(e_6 + e_8)
        X_t(row, k+2*m, j)   = ω_{8l}^{2(j-1)}(e_1 + e_3)
        X_t(row, k+3*m, j)   = ω_{8l}^{3(j-1)}(e_7 - e_9)
        X_t(row, k+4*m, j)   = ω_{8l}^{4(j-1)}(e_0 - e_2)
        X_t(row, k+5*m, j)   = ω_{8l}^{5(j-1)}(e_7 + e_9)
        X_t(row, k+6*m, j)   = ω_{8l}^{6(j-1)}(e_1 - e_3)
        X_t(row, k+7*m, j)   = ω_{8l}^{7(j-1)}(e_6 - e_8)
      end do
    end do
  end do
end do
```

図 2　基数 8 の DIF Stockham アルゴリズムに基づく ns 組の n 点 multirow FFT

☆ "-nolimit" は高度な最適化を行う場合，コンパイル時間およびコンパイル時に使用するメモリ量に制約を設けないオプションであり，"-W0,'opt(o(ss))'" は実行速度が最も速くなるような最適化を行うオプションである．

表 2　four-step FFT に基づく並列一次元 FFT の性能（HITACHI SR8000）

N	$P=1$		$P=2$		$P=4$		$P=8$		$P=16$	
	Time	GFLOPS	Time	GFLOPS	Time	GFLOPS	Time	GFLOPS	Time	GFLOPS
2^{20}	0.03086	3.398	0.01969	5.324	0.01279	8.199	0.00707	14.837	0.00486	21.562
2^{21}	0.06431	3.424	0.04027	5.469	0.02610	8.435	0.01362	16.169	0.00862	25.542
2^{22}	0.13102	3.521	0.08296	5.561	0.05161	8.939	0.02732	16.890	0.01609	28.667
2^{23}	0.27231	3.543	0.16804	5.741	0.10816	8.919	0.05495	17.557	0.03107	31.045
2^{24}	0.56249	3.579	0.34423	5.849	0.21955	9.170	0.11125	18.097	0.06287	33.022
2^{25}	1.20509	3.480	0.72733	5.767	0.44407	9.445	0.22894	18.321	0.12718	32.981
2^{26}	2.59680	3.360	1.54041	5.664	0.95271	9.157	0.47999	18.176	0.25888	33.699

表 3　five-step FFT に基づく並列一次元 FFT の性能（HITACHI SR8000）

N	$P=1$		$P=2$		$P=4$		$P=8$		$P=16$	
	Time	GFLOPS	Time	GFLOPS	Time	GFLOPS	Time	GFLOPS	Time	GFLOPS
2^{20}	0.03020	3.472	0.01938	5.412	0.01236	8.482	0.00696	15.065	0.00470	22.298
2^{21}	0.06310	3.490	0.04020	5.477	0.02524	8.725	0.01341	16.426	0.00821	26.806
2^{22}	0.12757	3.616	0.07946	5.806	0.04927	9.365	0.02624	17.580	0.01509	30.571
2^{23}	0.26185	3.684	0.16186	5.960	0.10221	9.439	0.05230	18.446	0.02921	33.026
2^{24}	0.53828	3.740	0.33068	6.088	0.21055	9.562	0.10537	19.106	0.05794	34.747
2^{25}	1.08061	3.881	0.66240	6.332	0.42840	9.791	0.21000	19.973	0.11440	36.665
2^{26}	2.19543	3.974	1.34259	6.498	0.83866	10.403	0.42365	20.593	0.22823	38.226

表 4　全対全通信の性能（HITACHI SR8000）

N	$P=2$		$P=4$		$P=8$		$P=16$	
	Time	MB/sec	Time	MB/sec	Time	MB/sec	Time	MB/sec
2^{20}	0.00488	859.38	0.00382	823.00	0.00301	608.93	0.00223	441.26
2^{21}	0.00955	878.24	0.00808	778.92	0.00559	656.92	0.00357	550.49
2^{22}	0.01887	889.16	0.01724	729.88	0.01071	685.24	0.00616	638.54
2^{23}	0.03748	895.24	0.03629	693.49	0.02092	701.83	0.01140	690.07
2^{24}	0.07468	898.57	0.07469	673.83	0.04137	709.76	0.02163	727.09
2^{25}	0.14908	900.28	0.15320	657.08	0.08162	719.46	0.04215	746.35
2^{26}	0.29784	901.26	0.30984	649.78	0.16252	722.62	0.08299	758.07

four-step FFT および five-step FFT に基づく N 点の並列一次元 FFT の性能を表 **2**，表 **3** に示す．ここで，実行時間の単位は秒であり，$N=2^m$ 点 FFT の GFLOPS 値は，$5N\log_2 N$ より算出している．

表 2 および表 3 から分かるように，five-step FFT に基づく並列一次元 FFT アルゴリズムが，four-step FFT に基づく並列一次元 FFT アルゴリズムに比べて高い性能が得られていることが分かる．これは，各ノード内において five-step FFT の方が four-step FFT に比べて最内側ループ長が長くなっているためであると考えられる．

表 3 から分かるように，16 ノードの SR8000 では five-step FFT に基づく並列一次元 FFT アルゴリズムにおいて，$N=2^{26}$ の場合に約 38 GFLOPS の性能が得られていることが分かる．

また，並列一次元 FFT の実行時間における通信時間の割合を調べるために，倍精度複素数の配列に対して全対全通信を 10 回実行し，その平均の経過時間を測定した．表 4 に，SR8000 における全対全通信の性能を示す．ここで，実行時間の単位は秒であり，通信性能（MB/sec）は全対全通信の通信量 $(P-1)\times(16N/P^2)$（バイト）より算出している．

表 3 によると，five-step FFT に基づく並列一次元 FFT アルゴリズムでは，16 ノードにおける $N=2^{26}$ 点 FFT の実行時間は 0.22823 秒となっているが，表 4 より全対全通信に要する時間は 0.08299 秒となっており，実行時間の 1/3 以上が通信時間であることが分かる．

ここで，従来の並列一次元 FFT アルゴリズム [4),6)] のように，入力と出力を同じデータ配置にした場合に全対全通信が 2 回必要である手法を用いたとすると，実行時間の半分以上が通信時間で占められることになると予想される．したがって，本論文で提案した全対全通信の回数を 1 回に削減する手法は実行時間を短縮する上で大きな効果があるといえる．

8. ま と め

本論文では，ベクトル SMP ノードを搭載した分散メモリ型並列計算機における並列一次元 FFT アルゴリズムを提案した．

従来の four-step FFT アルゴリズムにおいて，さら

に最内側ループ長を長くするために，本論文では five-step FFT を提案し，four-step FFT および five-step FFT アルゴリズムに基づき二種類の並列 FFT アルゴリズムを構成できることを示した．

本論文で提案した並列一次元 FFT アルゴリズムでは，データ配置にサイクリック分割を適用することにより，入力と出力を同じデータ配置にした場合でも全対全通信を 1 回で済ませることができた．

ベクトル SMP ノード内における FFT アルゴリズムでは，基数 4 および基数 8 の FFT を用いることにより，基数 2 の FFT に比べてノード内のメモリアクセス回数を減らすことができた．さらに，ベクトル SMP ノードでは，最内側ループ長だけでなく，外側ループ長も考慮する必要があることも示した．

提案するアルゴリズムに基づいて，並列一次元 FFT を（疑似）ベクトルノードを搭載した分散メモリ型並列計算機 HITACHI SR8000 に実現し，性能評価を行った．その結果，16 ノードの SR8000 では約 38 GFLOPS の性能を得ることができた．

今後の課題としては，データ数 N が $N = 2^p3^q5^r$ と表される場合について従来の FFT アルゴリズムよりも演算量の少ない GPFA (generalized prime factor FFT algorithm) [21] をベクトル SMP ノードを搭載した分散メモリ型並列計算機上に実現し，評価することがあげられる．

謝辞　本研究の一部は，文部省科学研究費補助金奨励研究（A）（課題番号 10780166）の支援を受けた．

参考文献

1) Cooley, J. W. and Tukey, J. W.: An Algorithm for the Machine Calculation of Complex Fourier Series, *Math. Comput.*, Vol. 19, pp. 297–301 (1965).
2) Swarztrauber, P. N.: Multiprocessor FFTs, *Parallel Computing*, Vol. 5, pp. 197–210 (1987).
3) Johnsson, S. L. and Krawitz, R. L.: Cooley-Tukey FFT on the Connection Machine, *Parallel Computing*, Vol. 18, pp. 1201–1221 (1992).
4) Agarwal, R. C., Gustavson, F. G. and Zubair, M.: A High Performance Parallel Algorithm for 1-D FFT, *Proc. Supercomputing '94*, pp. 34–40 (1994).
5) Hegland, M.: Real and Complex Fast Fourier Transforms on the Fujitsu VPP 500, *Parallel Computing*, Vol. 22, pp. 539–553 (1996).
6) 高橋大介, 金田康正: 分散メモリ型並列計算機による 2, 3, 5 基底一次元 FFT の実現と評価, 情報処理学会論文誌, Vol. 39, pp. 519–528 (1998).
7) Edelman, A., McCorquodale, P. and Toledo, S.: The Future Fast Fourier Transform?, *SIAM J. Sci. Comput.*, Vol. 20, pp. 1094–1114 (1999).
8) IBM Corporation: *Parallel Engineering and Scientific Subroutine Library Version 2 Release 1.2 Guide and Reference (SA22-7273)*, 3rd edition (1999).
9) Bailey, D. H.: FFTs in External or Hierarchical Memory, *The J. Supercomputing*, Vol. 4, pp. 23–35 (1990).
10) Greengard, L. and Gropp, W. D.: A Parallel Version of the Fast Multipole Method, *Comput. Math. Applic.*, Vol. 20, pp. 63–71 (1990).
11) Katzenelson, J.: Computational Structure of the N-Body Problem, *SIAM J. Sci. Stat. Comput.*, Vol. 10, pp. 787–815 (1989).
12) Nakazawa, K., Nakamura, H., Boku, T., Nakata, I. and Yamashita, Y.: CP-PACS: A Massively Parallel Processor at the University of Tsukuba, *Parallel Computing*, Vol. 25, pp. 1635–1661 (1999).
13) Brigham, E. O.: *The Fast Fourier Transform and its Applications*, Prentice-Hall, Englewood Cliffs, NJ (1988).
14) Van Loan, C.: *Computational Frameworks for the Fast Fourier Transform*, SIAM Press, Philadelphia, PA (1992).
15) Wadleigh, K. R., Gostin, G. B. and Liu, J.: High-Performance FFT Algorithms for the Convex C4/XA Supercomputer, *The J. Supercomputing*, Vol. 9, pp. 163–178 (1995).
16) Cochrane, W. T., Cooley, J. W., Favin, D. L., Helms, H. D., Kaenel, R. A., Lang, W. W., Maling, Jr., G. C., Nelson, D. E., Rader, C. M. and Welch, P. D.: What is the Fast Fourier Transform?, *IEEE Trans. Audio Electroacoust.*, Vol. 15, pp. 45–55 (1967).
17) Swarztrauber, P. N.: FFT Algorithms for Vector Computers, *Parallel Computing*, Vol. 1, pp. 45–63 (1984).
18) Bergland, G. D.: A Fast Fourier Transform Algorithm Using Base 8 Iterations, *Math. Comput.*, Vol. 22, pp. 275–279 (1968).
19) Goedecker, S.: Fast Radix 2, 3, 4, and 5 Kernels for Fast Fourier Transformations on Computers with Overlapping Multiply-Add Instructions, *SIAM J. Sci. Comput.*, Vol. 18, pp. 1605–1611 (1997).
20) 高橋大介, 金田康正: 積和演算に向いた 8 基底 FFT Kernel の提案, 情報処理学会研究報告 99-HPC-76, pp. 55–60 (1999).
21) Temperton, C.: A Generalized Prime Factor FFT Algorithm for Any $N = 2^p3^q5^r$, *SIAM J. Sci. Stat. Comput.*, Vol. 13, pp. 676–686 (1992).

【受入日】 2001年01月17日
【特許庁受入日】
【ＣＳターム】 BZ01、BZ02、BZ03、BZ04、CC01、CZ01、DZ02、DZ04、FZ01、GZ01、GG03、GZ03、GZ11、JJ05、JZ05、LZ08
【フリーワード】 分散メモリ、並列計算機、並列ＦＦＴ、アルゴリズム、ベクトルＳＭＰ、性能評価、高速Ｆｏｕｒｉｅｒ変換、Ｃｏｏｌｅｙ－Ｔｕｋｅｙアルゴリズム、コンパイラ、メモリアクセス回数、スカラープロセッサ、ベクトルプロセッサ、並列一次元ＦＦＴ、ＲＩＳＣプロセッサ、メモリ、多次元クロスバネットワーク、通信速度、離散Ｆｏｕｒｉｅｒ変換、行列、Ｓｔｏｃｋｈａｍアルゴリズム、ｍｕｌｔｉｒｏｗ ＦＦＴアルゴリズム、HITACHI SR8000、SMP、fast multipole method、Four-Step FFT、DFT、Five-Step FFT、FORTRAN、FORTRAN77、GPFA
【許諾レベル】 1Z
【著者群】
【著者名】 高橋 大介 DAISUKE TAKAHASHI
【著者所属】 東京大学 情報基盤センター
【論文タイトル】 分散メモリ型並列計算機ＨＩＴＡＣＨＩ ＳＲ８０００における並列ＦＦＴアルゴリズム
Parallel FFT Algorithms for the Distributed-Memory Parallel Computer HITACHI SR8000
【資料タイプ】 学術論文（国内）
【ジャーナル】
【ジャーナルタイトル】 ２０００年記念並列処理シンポジウム Ｖｏｌ．２０００ Ｎｏ．６ Joint Symposium on Parallel Processing 2000
【サブタイトル】 情報処理学会シンポジウムシリーズ IPSJ Symposium Series
【発行者名】 社団法人情報処理学会 Infrormation Processing Society of Japan
【開催日・発行日】 2000年06月01日
第2000巻 第6号
【頁】 91～98
【ISSN】 1344-0640